

```

%=====
%  PROGRAM steady.m
%=====

clear

%-----
%  LOAD DATA
%-----

load MESHo    -ASCII
load NODES    -ASCII
load NP        -ASCII
load NWLD     -ASCII

NUMNP = MESHo(1);
NUMEL = MESHo(2);
NNPE  = MESHo(3);

UX = linspace(0,0,NUMNP);
UY = linspace(0,0,NUMNP);

for I=1:NUMNP;
    XORD(I)=NODES(I,1);
    YORD(I)=NODES(I,2);
    NPcode(I)=NODES(I,3);
end

% -----
% Determine bandwidth and
% diagonal column for
% nonsymmetric storage
% -----
IB    = NWLD(NUMNP+1);
IDIAG=IB;
IB    = 2*IB-1;

% -----
% General Initialization
% -----
PI=4.0*atan(1.0);
if NNPE == 3
    NSPE=3;
    NNPS=2;
elseif NNPE == 6
    NSPE=3;
    NNPS=3;
elseif NNPE == 4
    NSPE=4;
    NNPS=2;
elseif NNPE == 8
    NSPE=4;
    NNPS=3;
end

for I=1:NUMNP;
    Q(I)=0;
    QS(I)=0;
    HS(I)=0;
    PSI(I)=0;

```

```

    PS1a(I)=0;
    NPBC(I)=0;
end

for I=1:NUMNP
    for J=1:IB
        SK(I,J)=0;
    end
end

% -----
% Ask User for type of analysis desired
% -----
%     disp(' ')
%     disp('ENTER:')
%     disp('-----')
%     disp('0 for 2D rectangular analysis')
%     disp('1 for axi-symmetric analysis about x-axis')
%     disp('2 for axi-symmetric analysis about y-axis')
%     disp('-----')
%     IRZ = input(' < ');

IRZ = 0;

if IRZ < 0 | IRZ > 2
    disp('-----')
    disp('ERROR IN INPUT ')
    disp('IRZ has been set to 0')
    disp('-----')
    IRZ = 0;
end

% -----
% Get shape function quadrature data
% -----
[ SF,WT,NUMQPT,NPSIDE] = SFquad(NNPE);

% -----
% Include user written initialization
% -----
velocity_INITIAL

%-----
% Place Q in RHS, compact storage
%-----
for I=1:NUMNP
    RHS (NWLD(I))=Q(I);
end

% -----
% Create element matrices
% -----
for I=1:NUMEL;
    LMENT=I;
    for J=1:NNPE;
        QE(J)=0.0;
    end
end

```

```

    for K=1:NNPE;
        S(J,K)=0;
    end
end

% -----
% Begin volume quadrature for each element
% -----
JEND=NUMQPT(1);
for J=1:JEND;
    XJ=0;
    YJ=0;
    RJAC(1,1)=0;
    RJAC(1,2)=0;
    RJAC(2,1)=0;
    RJAC(2,2)=0;

% -----
% Determine coordinate and Jacobian
% -----
    for K=1:NNPE;
        NPK=NP(I,K);
        XJ=XJ+SF(1,K,J)*XORD(NPK);
        YJ=YJ+SF(1,K,J)*YORD(NPK);
        RJAC(1,1)=RJAC(1,1)+SF(2,K,J)*XORD(NPK);
        RJAC(1,2)=RJAC(1,2)+SF(3,K,J)*XORD(NPK);
        RJAC(2,1)=RJAC(2,1)+SF(2,K,J)*YORD(NPK);
        RJAC(2,2)=RJAC(2,2)+SF(3,K,J)*YORD(NPK);
    end

    DETJ=RJAC(1,1)*RJAC(2,2)...
        -RJAC(2,1)*RJAC(1,2);
    if DETJ <= 0
        fprintf(1, '\n-----')
        fprintf(1, '\n Error in steady.m      ')
        fprintf(1, '\n DETJ =%7e', DETJ        )
        fprintf(1, '\n must be > 0.0'          )
        fprintf(1, '\n-----\n')
        error
    end

% -----
% Determine inverse of Jacobian
% -----
    RJACI(1,1)=+RJAC(2,2)/DETJ;
    RJACI(1,2)=-RJAC(1,2)/DETJ;
    RJACI(2,1)=-RJAC(2,1)/DETJ;
    RJACI(2,2)=+RJAC(1,1)/DETJ;

% -----
% Determine derivative of shape functions in X-Y plane
% -----
    for K=1:NNPE;
        DNDX(K)=RJACI(1,1)*SF(2,K,J)+RJACI(2,1)*SF(3,K,J);
        DNDY(K)=RJACI(1,2)*SF(2,K,J)+RJACI(2,2)*SF(3,K,J);
    end
end

```

```

% -----
% Include user written coefficients
% RXJ, RYJ, BXJ, BYJ, GVJ, HVJ
% -----
velocity_COEF

% -----
% Create element stiffness matrix
% -----
DV=DETJ;
if IRZ == 1
    DV=2.0*PI*XJ*DV;
elseif IRZ == 2
    DV=2.0*PI*YJ*DV;
end

for K=1:NNPE;
    QE(K)=QE(K)+WT(1,J)*SF(1,K,J)*HVJ*DV;
    SFK=SF(1,K,J);
    for L=1:NNPE;
        SFL=SF(1,L,J);
        S(K,L)=S(K,L)+ ...
            WT(1,J)*(DNDX(K)*RXJ*DNDX(L)+DNDY(K)*RYJ*DNDY(L) ...
                -SFK*BXJ*DNDX(L)-SFK*BYJ*DNDY(L) ...
                -SFK*GVJ*SFL)*DV;
    end
end
end
% ----- end of volume quadrature

% -----
% Begin surface quadrature
% -----
for J=1:NSPE
    CHKH=1;
    CHKQ=1;
    for K=1:NNPS;
        J1=NP(I,NPSIDE(J,K));
        CHKH=CHKH*HS(J1);
        CHKQ=CHKQ*QS(J1);
    end

    if CHKH ~= 0 | CHKQ ~= 0
        KEND=NUMQPT(2);
        for K=1:KEND;
            YK = 0.0;
            XK = 0.0;
            dxdu = 0.0;
            dydu = 0.0;
            HSK = 0.0;
            QSK = 0.0;
            PSIK = 0.0;
            for L=1:NNPS;
                L1=NPSIDE(J,L);
                NPL=NP(I,L1);
            end
        end
    end
end

```

```

        XK  =XK  +SF(4,L,K)*XORD(NPL);
        YK  =YK  +SF(4,L,K)*YORD(NPL);
        dxdu =dxdu +SF(5,L,K)*XORD(NPL);
        dydu =dydu +SF(5,L,K)*YORD(NPL);
        QSK  =QSK  +SF(4,L,K)*QS(NPL);
        HSK  =HSK  +SF(4,L,K)*HS(NPL);
        PSIK =PSIK +SF(4,L,K)*PSIa(NPL);
    end
    dsdu = sqrt(dxdu ^2+dydu ^2);
    duds = 1/dsdu;

    DS=dsdu;
    if IRZ == 1
        DS=2.0*PI*XK*DS;
    elseif IRZ == 2
        DS=2.0*PI*YK*DS;
    end

    for L=1:NNPS;
        L1=NPSIDE(J,L);
        if CHKQ ~= 0
            QE(L1)=QE(L1)+WT(2,K)*SF(4,L,K)*QSK*DS;
        end
        if CHKH ~= 0
            QE(L1)=QE(L1)+WT(2,K)*SF(4,L,K)*HSK*PSIK*DS;
            for M=1:NNPS;
                M1=NPSIDE(J,M);
                S(L1,M1)=S(L1,M1)+...
                    WT(2,K)*SF(4,L,K)*HSK*SF(4,M,K)*DS;
            end
        end
    end

    end % of surface quadrature

    end % if-statement for quadrature

end % of loop over element sides

% -----
% Place completed element matrix in global SK and Q matrices
% -----
for J=1:NNPE
    JNP=NP(I,J);
    JEQ=NWLD(JNP);
    RHS(JEQ)=RHS(JEQ)+QE(J);
    for K=1:NNPE
        KNP=NP(I,K);
        KEQ=NWLD(KNP);
        KB=(KEQ-JEQ)+IDIAG;
        SK(JEQ,KB)=SK(JEQ,KB)+S(J,K);
    end
end

end % Loop over elements

```

```
% -----  
% Specify known values of PSI  
% -----  
for I=1:NUMNP;  
    if NPBC(I) == 1  
        NI=NWLD(I);  
        SK(NI,IDIAG)=SK(NI,IDIAG)*1.0E+06;  
        RHS(NI)=PSI(I)*SK(NI,IDIAG);  
    end  
end  
  
% -----  
% Call Gauss elimination routine  
% -----  
PSI = nGAUSS(SK,RHS,NUMNP,IB);  
  
% -----  
% Renumber solution values using original node numbers  
% -----  
for I=1:NUMNP;  
    NI=NWLD(I);  
    RHS(I)=PSI(NI);  
end  
for I=1:NUMNP  
    PSI(I)=RHS(I);  
end  
  
save PSI PSI -ASCII  
  
save UX UX -ASCII  
save UY UY -ASCII
```